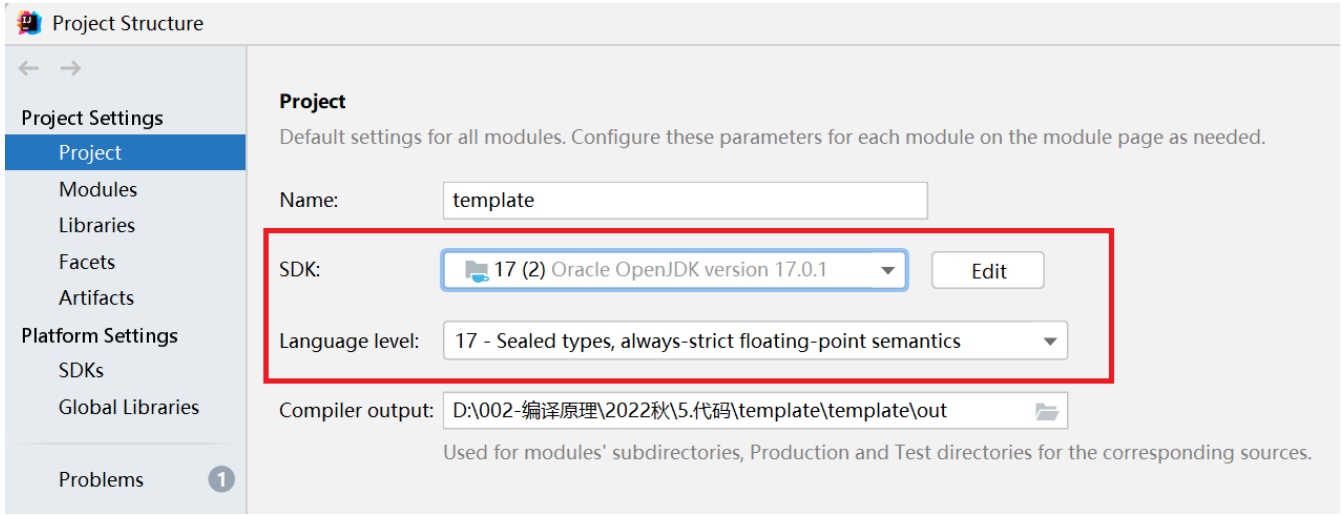
编译原理

实验一：词法分析

规格严格，功夫到家

编译原理实验安排

- **实验目标**
  - ✓ 实现一个编译器
  - ✓ 目标平台是RISC-V 32
- **实验语言和环境**
  - ✓ JAVA语言
  - ✓ IntelliJ IDEA
  - ✓ JDK17及以上版本



编译原理实验安排

框架代码下载地址

https://gitee.com/hitsz-cslab/Compiler/releases/tag/2022F.0.1

在线指导书

https://compiler-6bi.pages.dev/

编译原理实验安排

✔16个学时，完成4个实验；

实验	学时	提交	实验题目
实验一	2学时		词法分析
实验二	8学时	提交实验一代码	自底向上的语法分析（LR(1)）
实验三	4学时	提交实验二代码	典型语句的语义分析及中间代码生成
实验四	2学时	提交实验三代码	目标代码生成

备注：本学期编译原理四次实验，只需完成所有实验任务后提交一份完整实验报告。

实验报告

编译原理实验报告

编译原理实验安排

✔实验提交内容：

1. 电子版代码

电子版内容：工程文件

压缩文件命名：学号-姓名-实验x.zip

实验报告命名：学号-姓名-编译原理实验报告.pdf

✔实验总分数：20分。实验报告占30%，代码占70%

实验一	词法分析	12.5分(百分制)
实验二	自底向上的语法分析（LR(1)）	50分(百分制)
实验三	典型语句的语义分析及中间代码生成	25分(百分制)
实验四	代码生成	12.5分(百分制)

实验提交

实验提交网址：http://grader.tery.top:8001/

✔ 具体每次实验提交截止时间请参考作业系统上截止时间

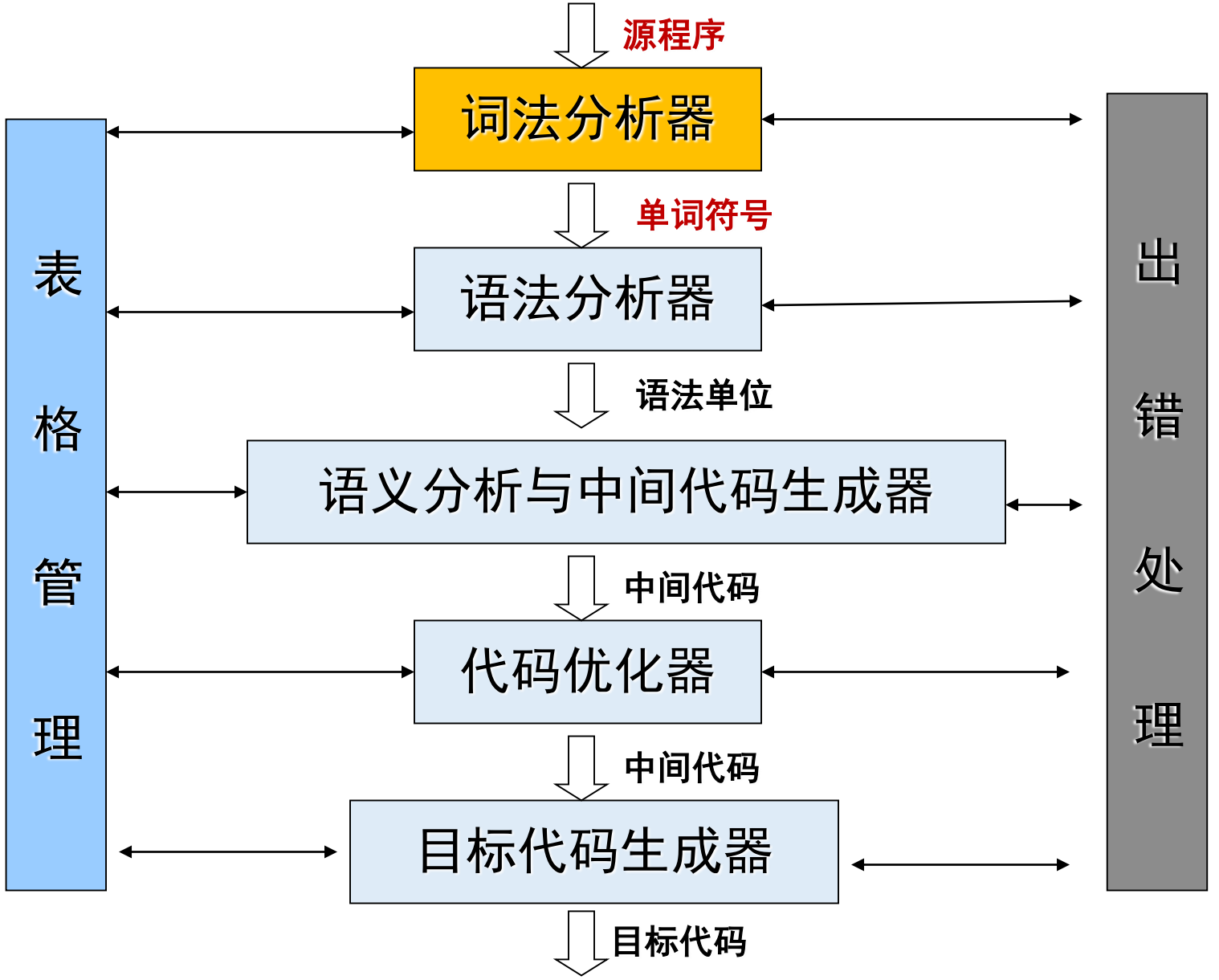
编译程序的总体结构

✔实验一：

✔实验二：

✔实验三：

✔实验四：



实验目的

1. 加深对词法分析程序的功能及实现方法的理解。

2. 对类C语言单词符号的文法描述有更深的认识，理解有限自动机、编码表和符号表在编译的整个过程中的应用。

3. 设计并编程实现一个词法分析程序，对类C语言源程序段进行词法分析，加深对高级语言的认识。

备注：类C语言指C语言子集或者自定义的其他类似C语言语法的编程语言；

实验学时数：2学时

实验内容

编写一个词法分析程序，读取文件，对文件内的类C语言程序段进行词法分析。

1. 输入：以文件形式存放的类C语言程序段；

```
data/in
├── coding_map.csv       # 码点文件
└── input_code.txt       # 输入的代码
```

2. 输出：以文件形式存放的TOKEN串和简单符号表；

```
data/out
├── old_symbol_table.txt     # 符号表
└── token.txt                # 词法单元列表
```

词法分析器的设计分析

1. 词法分析器的功能

对源程序进行编译预处理（去除注释、无用回车换行等）之后，把源程序分析成一个个单词。由此可知，词法分析器的输入是源语言字符流，输出是单词序列。

2. 词法分析器如何识单词

通过正则文法来描述单词的构成规则。

3. 编码表

编译器为了处理方便，按照一定的方式对单词进行分类和编码，所以需要定义一个编码表。

4. 编写程序

使用有限自动机DFA作为桥梁。

实验步骤

1. 定义编码表；（框架已提供）
2. 创建属于自己的类C语言单词符号的文法；
3. 根据所建文法画出有限自动机的状态转换图；
4. 定义数据结构：TOKEN串（已提供），符号表；
5. 根据有限自动机的状态转移编写代码；
6. 输出Token串和符号表到指定文件中；
7. 验证实验结果；
8. 完成实验报告的词法分析相关内容。

程序语言单词的种类

（1）标识符：由用户定义，表示各种名字；

（2）关键字：也称基本字，do、while、int、char、sizeof...

（3）常数：整常数、实常数、字符常量、字符串常量、符号常量；

（4）运算符：算术运算符+、-、*、/等；

逻辑运算符not、or与and等；

关系运算符=、<>、>=、<=、>和<等；

（5）分界符：，、；、（、）...；

定义编码表

单词名称	类别编码	单词值
int	1	–
return	2	–
=	3	–
,	4	–
Semicolon	5	–
+	6	–
–	7	–
*	8	–
/	9	–
(	10	–
)	11	–
id	51	内部字符串
IntConst	52	整数值
……	……	……
布尔常数	80	0 或 1
字符串常数	81	内部字符串

注意：

- 关键字、运算符、分界符一符一码
- 标识符一个编码
- 常量一类一码

} 使用单词值区分

文法设计

➢正则文法表示

G=(V,T,P,S)，其中V={S,A,B,Cdigit,no_0_digit,,char},T={任意符号}，P定义如下

约定：用digit表示数字：0,1,2,…,9; no_0_digit表示数字：1,2,…,9;

用letter表示字母：A,B,…,Z,a,b,…,z,_

标识符：S→letter A　　A→letter A|digit A|ε

整常数：S ->no_0_digitB　B -> digitB | ε

运算符：S -> C　　C -> =|*|+|-|/

➢正则表达式表示法

标识符：id→letter (letter|digit)*

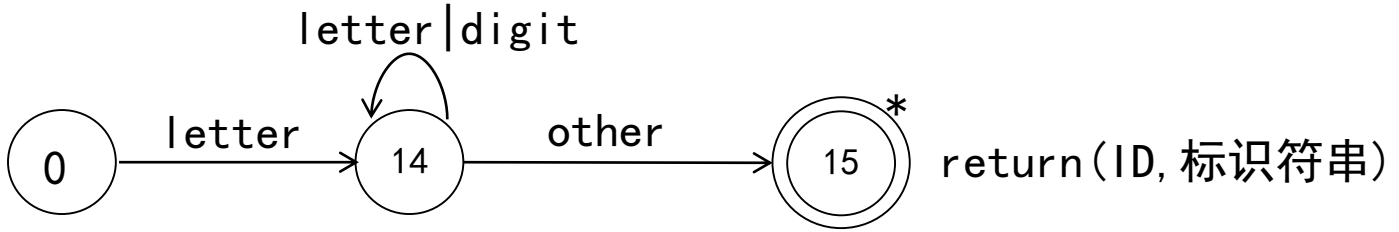
整常数：id→no_0_digit(digit|)*

有限自动机

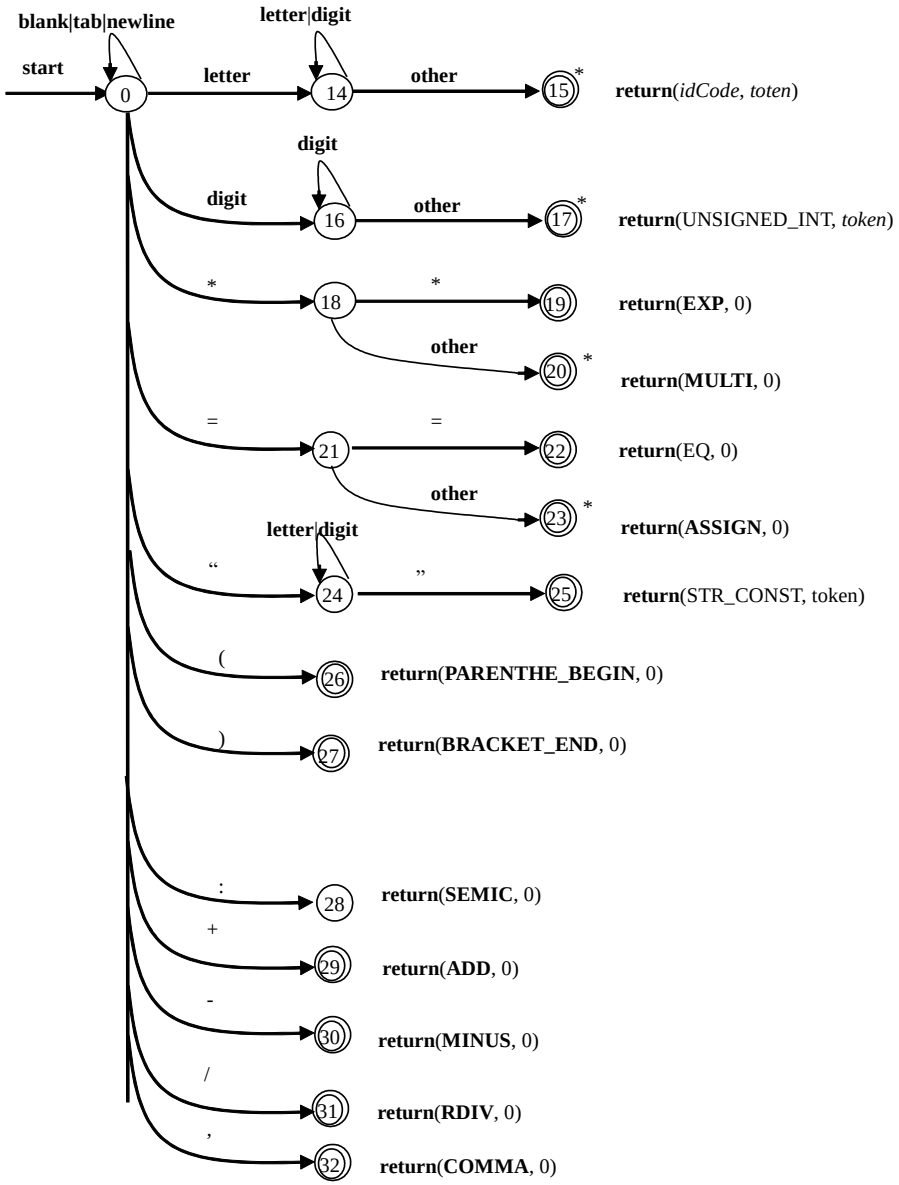
有限状态自动机和正则文法等价，考虑到状态转换图的直观性，我们从状态转换图出发来考虑词法分析器的设计。

标识符正则表达式：id→letter (letter|digit)*

识别标识符和关键字的状态转换图：



各类状态转换图合并


blank|tab|newline
letter|digit
start
letter
other
return(idCode, toten)
digit
digit
other
return(UNSIGNED_INT, token)
*
*
return(EXP, 0)
other
return(MULTI, 0)
=
=
return(EQ, 0)
other
return(ASSIGN, 0)
letter|digit
“
”
return(STR_CONST, token)
(
return(PARENTHE_BEGIN, 0)
)
return(BRACKET_END, 0)
:
return(SEMIC, 0)
+
return(ADD, 0)
-
return(MINUS, 0)
/
return(RDIV, 0)
,
return(COMMA, 0)

词法分析器输出——TOKEN串

举例

输入源代码：

```
int result;
int a;
int b;
int c;
a = 8;
b = 5;
c = 3 - a;
result = a * b - ( 3 + b )
* ( c - a );
return result;
```

词法分析输出token：

```
(int,)
(id,result)
(Semicolon,)
(int,)
(id,a)
(Semicolon,)
(int,)
(id,b)
(Semicolon,)
(int,)
(id,c)
(Semicolon,)
(id,a)
(=,)
(IntConst,8)
(Semicolon,)
(id,b)
(=,)
(IntConst,5)
(Semicolon,)
(id,c)
(=,)
(IntConst,3)
(-,)
(id,a)
(Semicolon,)
(id,result)
(=,)
(id,a)
(*,)
(id,b)
(-,)
((,)
(IntConst,3)
(+,)
(id,b)
(),)
(*,)
((,)
(id,c)
(-,)
(id,a)
(),)
(Semicolon,)
(return,)
(id,result)
(Semicolon,)
```

表示单词的种类，可用整数编码或记忆符表示

种别码	属性值

不同的单词不同的值

词法分析器的输出——符号表

符号表：以名字为关键字来记录其信息的数据结构；

支持的基本操作包括插入、查找和删除，本实验实现前两种即可。

符号表的数据结构类型：线性表（优点：简单直观，缺点：插入时间复杂度高）

散列表（查找、插入效率高）

	名字	属性
符号表表项1	abc	...
符号表表项2	i	...
⋮	...	...
符号表表项*n*	...	...

图：符号表

框架代码主程序流程说明


```
final var lexer = new LexicalAnalyzer(symbolTable);
lexer.loadFile(FilePathConfig.SRC_CODE_PATH);
lexer.run();
lexer.dumpTokens(FilePathConfig.TOKEN_PATH);
```

```
final var parser = new SyntaxAnalyzer(symbolTable);
parser.loadTokens(tokens);
parser.loadLRTable(lrTable);
```

```
final var productionCollector =
new ProductionCollector(GrammarInfo.getBeginProduction());
parser.registerObserver(productionCollector);
```

whenShift(…)：进栈

whenReduce(…)：语义分析，更新符号表

```
final var semanticAnalyzer = new SemanticAnalyzer();
parser.registerObserver(semanticAnalyzer);
```

whenShift(…)：进栈

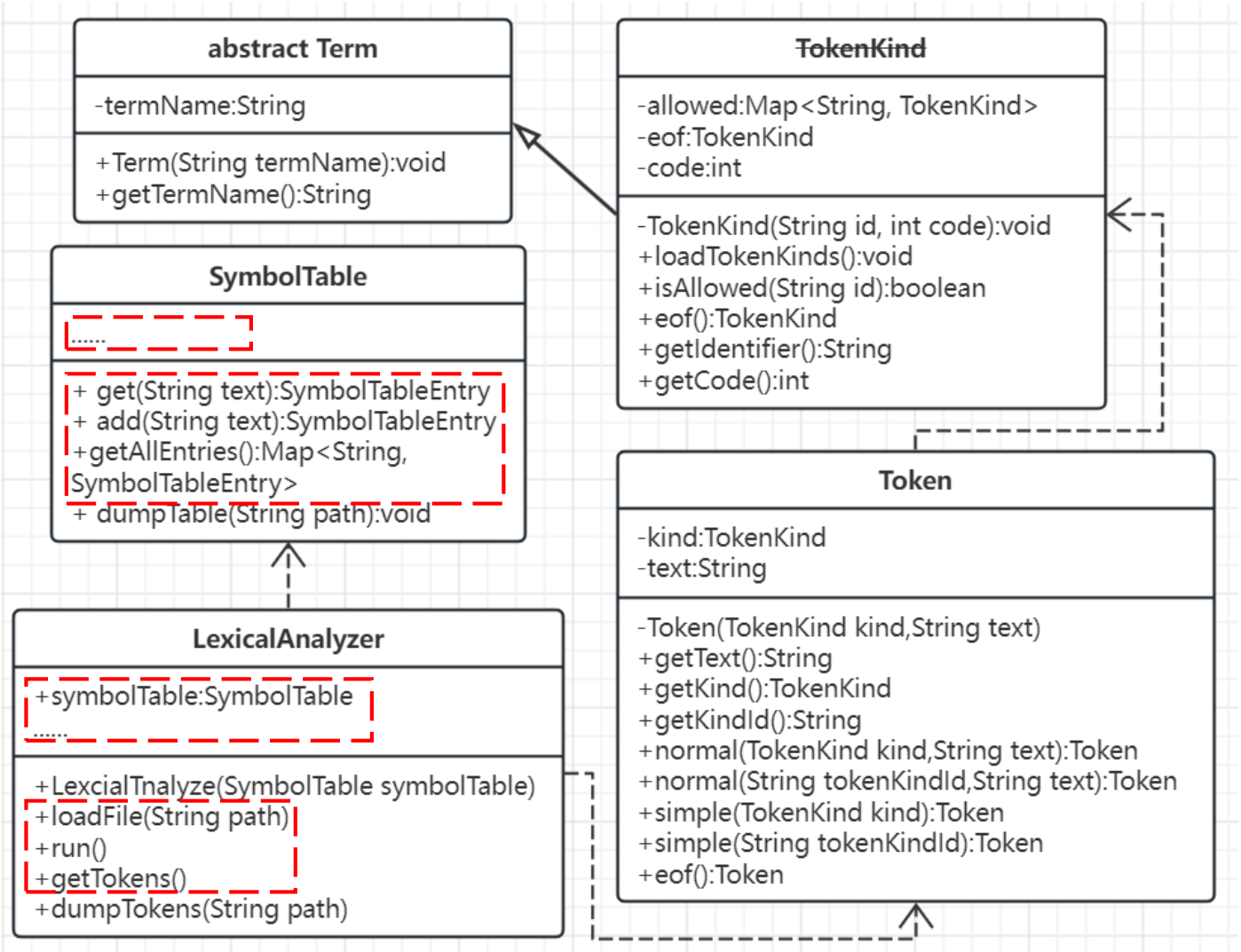
whenReduce(…)：编写语义动作，生成中间代码

```
parser.run();
```

```
final var asmGenerator = new AssemblyGenerator();
asmGenerator.loadIR(instructions);
asmGenerator.run();
```

```
productionCollector.dumpToFile(FilePathConfig.PARSER_PATH);
symbolTable.dumpTable(FilePathConfig.NEW_SYMBOL_TABLE);
irGenerator.dumpIR(FilePathConfig.INTERMEDIATE_CODE_PATH);
```

词法分析程序类图


abstract Term
-termName:String
+Term(String termName):void
+getTermName():String
TokenKind
-allowed:Map<String, TokenKind>
-eof:TokenKind
-code:int
-TokenKind(String id, int code):void
+loadTokenKinds():void
+isAllowed(String id):boolean
+eof():TokenKind
+getIdentifier():String
+getCode():int
SymbolTable
.....
+ get(String text):SymbolTableEntry
+ add(String text):SymbolTableEntry
+getAllEntries():Map<String, SymbolTableEntry>
+ dumpTable(String path):void
Token
-kind:TokenKind
-text:String
-Token(TokenKind kind,String text)
+getText():String
+getKind():TokenKind
+getKindId():String
+normal(TokenKind kind,String text):Token
+normal(String tokenKindId,String text):Token
+simple(TokenKind kind):Token
+simple(String tokenKindId):Token
+eof():Token
LexicalAnalyzer
+symbolTable:SymbolTable
.....
+LexcialTnalyze(SymbolTable symbolTable)
+loadFile(String path)
+run()
+getTokens()
+dumpTokens(String path)

验证实验结果

```
Terminal  Local
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

PS D:\project\CompilerProject\compiler-course-dev-SimpleAsm\compiler-course> python -u ./scripts/diff.py ./data/out/token.txt ./data/std/token.txt
The src file is the same as std file.
PS D:\project\CompilerProject\compiler-course-dev-SimpleAsm\compiler-course> python -u ./scripts/check-result.py 1 ./data/out ./data/std
Diffing lab1 output:
Diffing file token.txt:
The src file is the same as std file.
Diffing file old_symbol_table.txt:
The src file is the same as std file.
```

比对单个文件内容

比对前N个实验的所有输出

说明：

1. 框架模板仅支持声明语句、赋值语句和简单算术表达式。如同学想开发的编译器支持更多语法功能需自行修改以下文件：

➢源代码文件：input_code.txt；

➢编码表文件：coding_map.csv；

如实现了更多功能请在实验报告中注明

2. 注意事项：

➢禁止使用java.util提供的正则表达式的文本处理工具来完成词法分析；

➢禁止直接查找空格来分割单词；

同学们，
独立开始实验